(2)

The Counselor Project

Department of Computer and Information Science

University of Massachusetts

Amherst, Mass. 01003

Case-Based Reasoning as a Paradigm for Heuristic Search

Wendy G. Lehnert
October, 1987
CPTM #20

This series of internal memos describes research in artificial intelligence conducted under

DARPA CONTRACT N00014-87-K-0238

Case-Based Reasoning as a Paradigm for Heuristic Search

Wendy G. Lehnert

October, 1987
CPTM #20

Case-Based Reasoning as a Paradigm for Heuristic Search

Wendy G. Lehnert
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003
CSNET: LEHNERT.UMASS@CSNET-RELAY

ABSTRACT

Case-based reasoning (CBR) systems utilize known solutions to specific problems (a case base) in order to guide problem solving behavior in the face of new problems. The CBR approach to problem solving can easily be exploited to achieve learning capabilities if we incorporate new knowledge into the case base as we acquire additional problem-solving experience. In this paper we will show how the case-based reasoning approach to problem solving can be used to heuristically limit exhaustive tree searches. As an important aspect of this approach, we will also show how the concept of "coarse indices" can be used to expand the power of specific problem/solution pairs to order to saturate a search space. We have applied these ideas to an implementation of the 8-puzzle in an effort to illustrate how case-based reasoning fosters a class of weak methods for general problem solving and machine learning.

1 Introduction

In recent years, a new paradigm for problem solving and learning has begun to emerge which relies on accessing specific solutions to specific problems in order to handle new problem situations. Case-based problem solving and case-based learning algorithms are currently being investigated for a variety of tasks including general problem solving [Kolodner, Simpson, & Sycara-Cyranski, 1985], legal reasoning [Rissland, 1983; Ashley, 1987], medical diagnosis [Bariess & Porter, 1987], adaptive planning [Alterman, R., 1986], opportunistic planning [Hammond, 1986a], failure-driven learning [Hammond, 1986b], conflict negotiations [Simpson, 1986; Sycara, 1987], and word pronunciation [Lehnert, 1987]. A closely related effort inspired by massively parallel computer architectures is memory-driven reasoning (MBR) [Stanfill & Waltz, 1986] which utilizes stochastic information about large-scale knowledge bases. In all cases, the CBR (or MBR) technologies potentially support learning algorithms as a natural side-effect of their problem-solving strategies, and deserve further investigation as a paradigm for machine learning.

Because the CBR effort is relatively new, no public effort has been made to standardize the terminology associated with this technology or identify its standard techniques. In this paper we will attempt to describe the "bare-bones" essence of a CBR system by applying the approach to the 8-puzzle, a task that has been characterized in terms of heuristic search [Nilsson, 1971], problem similarity [Gaschnig, 1979], macro operators [Korf, 1985], preference predicates [Utgoff, 1987], and chunking mechanisms [Laird, Newell, & Rosenbloom, 1987; Laird, Rosenbloom, &

Newell, 1984; Laird, Rosenbloom, & Newell, 1986]. Our CBR approach to the 8-puzzle allows us to solve the puzzle for arbitrary boards, given a small case base of known solutions for a random set of problem boards.

Before we begin with a description of this system, a few words about the 8-puzzle as a CBR task are in order. One of the standard claims associated with CBR systems is a claim about explanatory power. Case-based reasoning systems are supposed to provide intelligible explanations for their behavior because they purport to reason as people do in situations characterized by incomplete information and non-optimal solutions. Most CBR systems operate in domains where the notion of an "intelligible" explanation is assumed to be an explanation that satisfies users in a position to ask questions. In general, these explanations are typically formulated in terms of remindings, near-miss matches, and analogical situations.

However convenient the 8-puzzle may be for illustrating heuristic search, it is a very poor problem with respect to intelligible explanation. For one thing, this is a puzzle where computers can easily outperform humans. There is no real advantage in trying to model a person solving the 8-puzzle. It follows that there is no pressing need for an 8-puzzle program that can defend itself in the face of a skeptical observer. If we had to design an interface for our program that would field the question "why did you move there?" we could do little more than identify a previously encountered solution to the 8-puzzle that is deemed relevant to the situation at hand. Unfortunately, a reference to another 8-puzzle solution is a rather weak excuse for an intelligible explanation.

One other problem with the 8-puzzle that limits its generality is the fact that the 8-puzzle embodies complete knowledge. There are no random factors or uncontrollable variables that enter into the problem. It follows from this that no interesting run-time problem solving can take place. If we can find a path of moves that will take us to the desired final state, there is nothing to stop us from achieving the final state. Most "real-world" problem solving domains are not so agreeable. Although this limitation is serious, run-time problem solving is less central to the CBR paradigm than the issue of explanation.

With these two caveats in mind, we will now present our 8-puzzle program as an imperfect prototype for general CBR programs. In spite of its shortcomings, many aspects of CBR technology will find natural expressions in addressing the 8-puzzle. For this reason, we feel justified in presenting our CBR approach to the 8-puzzle as a noteworthy exercise. By applying CBR techniques to a simple puzzle that does not appear to lend itself to the approach, we hope to argue for the generality of CBR research.

2 The Basic 8-Puzzle Program

Our general goal is to design a program that can generate solutions to the 8-puzzle by examining previously encountered solutions to the 8-puzzle. For our purposes, it will suffice to think of a solution as a sequence of legal moves that takes us from a given problem state to a targeted final state (see Figure 1). The first crucial question we must resolve concerns the representation

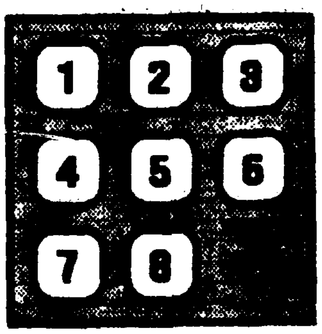

Figure 1: The Final State

of these sequences. We could (1) choose to represent the resulting boards associated with a move sequence, or we might (2) represent the moves themselves without the boards. In either case, we have to worry about the generality of our representation.

2.1 Representing the Case Base

First we must recognize an unavoidable trade-off between the generality of a representation and the difficulty of indexing useful solutions during problem solving. In the case of finite-table lookup, we sacrifice all claims for generality but benefit from reliable indices whenever a problem state is present in the case base. Unfortunately, a lack of generality will cripple any system where we have a large search space. The combinatorics of the 8-puzzle provide for 181,440 legal board positions. If our case base can only recognize boards that match previously encountered boards, we will have to amass a case base containing each of the 181,440 boards in order to insure that all problem states can be solved. While finite-table lookup qualifies as a degenerate form of CBR, it is not an acceptable solution for problems of any complexity.

The problem of accessing a case base is frequently described as an indexing problem. Any finite-table of solutions might be effectively utilized if we understand how to relate an arbitrary problem state to the entries available in the table. This amounts to knowing how to index the table effectively. Representations that incorporate a lot of generality are relatively easy to index, whereas representations that tend to be highly specific are more difficult to index.

One way to win both ways is to design an indexing technique that does not guarantee reliable results. For example, traditional hashing algorithms index a "bucket" of objects and fail to provide unique indices for all objects. Similarly, a partial pattern matcher is designed to relax similarities between structures at the risk of allowing inappropriate matches. We will use a technique for "coarse" indexing based on abstract features of board configurations in order to incorporate a lot of generality into the case base. In so doing, we understand that the solutions retrieved through the use of these indices are not always going to be useful for the problem at hand.

The specific indexing technique we use maps all legal 8-puzzle boards to 12 possible indices which are naturally represented by a set of 12 non-negative integers. To compute the index of a given board, we compare that board to the targeted final state by focusing on pairs of adjacent pieces within the two boards. Each 8-puzzle board has 3 rows and 3 columns, and each of these rows and columns contains 2 pairs of adjacent pieces. If we collect all the adjacent pairs derived from all rows and all columns, we find 12 pairs of 8-puzzle symbols.[1] We will view these pairs as sets rather than ordered pairs. We will call the set of all adjacent pairs associated with a board the "adjacency set" for that board.

Given a specific problem board, we can now compare that board's adjacency set to the adjacency set associated with the targeted final state. The comparison we make between these two adjacency sets can be thought of as a metric, describing a distance between the two boards. If our sets are equal, they differ by a distance equal to 0. If the two sets contain no shared elements, they differ by a distance of 12. If they share exactly 8 elements, we characterize their distance as 4. Note that it is possible for two boards to differ by all integral distances between 0 and 12 except for the value 1. It is not possible for two legal board configurations to share 11 pairs of adjacent pieces unless they share all 12 pairs. We will call this distance the "adjacency index" for problem boards.

By computing the adjacency index for any given board, we can represent all possible boards in terms of 12 metric equivalence classes. (Remember that we are holding the target state constant throughout these computations.) This is how our program will represent boards for the 8-puzzle. In order to enter a move sequence into the case base, we compute the adjacency index for each board associated with that sequence. We then add this list of integers to the case base and we're done.

2.2 Using the Case Base for Heuristic Search

Now suppose we have a case base of some 8-puzzle solutions encoded using the adjacency-index described above, and suppose we also have a problem state that we would like to transform into the final state. If we consider all possible paths of legal moves emanating from the initial problem board, we are dealing with a tree whose average branching factor is 2.67. An exhaustive traversal of the tree will eventually produce a sequence of moves that leads to the final state, but the search may go on for quite a while and there is nothing remotely intelligent about a brute force tree search.

We will improve on the brute force search by using our case base to restrict the tree traversal. To see how this works, imagine the complete tree emanating from the problem board with each node of the tree corresponding to a board configuration. We represent each of those boards using the adjacency-index. Each path in the tree starting at the root node is now a sequence of integers between 0 and 12 (not including 1). If we ever arrive at a node whose value is 0, we will terminate our search.

[1]Normally the symbols are the numbers 1,2,3,4,5,6,7,8, and a blank.

To initiate our tree search, we must first locate all instances of the root node index within the case base. Suppose the root node is indexed with a 9. If there are 100 instances of a 9 in the case base, we will now need to access each of those 100 positions in the case base. Note that these 100 positions also correspond to 100 paths in the case base beginning with a 9 and ending with a 0 since all cases in the case base end with 0.

Case Base Solutions:

(6 8 8 7 5 7 6 7 5 3 0)

(10 8 9 8 9 10 8 8 7 8 7 7 7 5 3 0)

(8 9 8 9 7 7 5 7 8 8 6 7 5 5 4 3 0)

Problem Search Space:

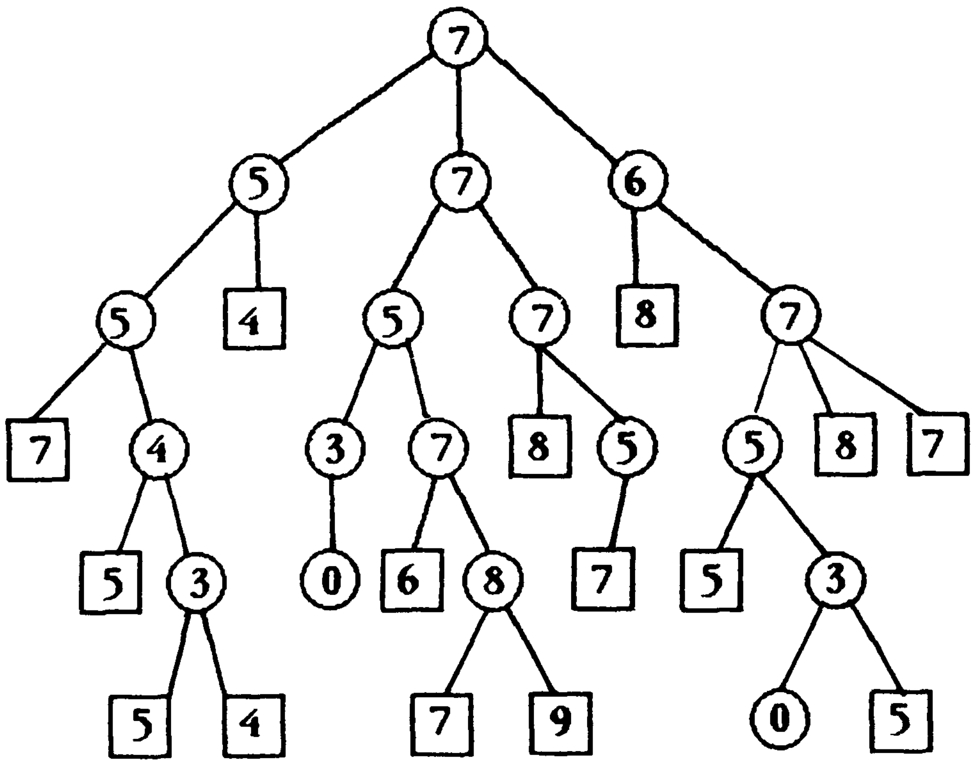

Figure 2: Limiting the Tree Search

We will now begin to traverse the first available pathway in the search space of legal 8-puzzle moves. Suppose the next node in this path has the value 12. Before we consider making this possible move, we will check it against the case base to see if any of the 100 pointers we have into our case base yields a path beginning with the sequence 9, 12. If no such path can be found, we will not pursue this branch of the search space any further. If at least one such path can be found, we will make the move and consider the possibilities available to us from there. By recursively evaluating all possible moves with the case base in hand, we can rule out any pathways that do not correspond to existing sequences in the case base. Figure 2 illustrates a truncated search space tree in the context of a case base containing 3 known solution paths. The square nodes indicate moves that are not consistent with any paths available in the case base. Note that this tree yields two possible solution paths which terminate at a node with value 0: one contains 4 moves and the other contains 5 moves.

In general, one of two things must happen when we apply this heuristic. (1) If a pathway is found in the search space that corresponds to a sequence in the case base ending with 0, then we have a solution to the problem state. (2) If no such pathway is found, the search will terminate without a solution. There is no way to know beforehand whether or not a given problem board can be solved using a specific case base. We will refer to this tree traversal algorithm as the "case base search." The leading index for a successful path terminating at 0 is called an "entry point" for the case base.

By recording the exact moves needed to traverse the search space tree, we can produce a specific move sequence in the event that the case base search is successful. Because the 0 index describes the rotation group of the final state, it is possible for a successful path to result in a rotation of the final state rather than the final state. If this happens, we can recognize the rotation on hand and apply a pre-defined move sequence to this board in order to produce the desired final state. This minor adjustment at the end of a successful case base search is unavoidable whenever the terminating index does not guarantee a unique board. In our case, we are dealing with only three possible adjustments[2] - a relatively small price to pay for representational generality.

2.3 Near-Miss and Far-Miss Solutions

If we apply the search procedure to an initial problem board and find a solution, we are done. But more often than not, we can expect this search to fail. The chances of matching an arbitrary board against a "small" case base are slight. However, this does not mean we cannot solve the problem. It may be possible to find a solution for a board configuration that is related to the initial board by a minor board modification. In that case, we would say that it is possible to solve the initial board on the basis of a "near-miss."

In our program, we have implemented a near-miss heuristic based on neighborhoods within the search space. Let us define the n-family of a board to be the set of all boards that can be obtained by applying n or less moves to the initial board. If the case base search fails to produce a solution for the initial board configuration, we then apply the case base search to each

[2]rotations by 90°, 180°, and 270°.

element of the initial board's 4-family until either (1) a solution is found, or (2) all elements of the 4-family are processed without success.[3] In the event that a solution is found within the 4-family, we say we have located a near-miss solution, or a solution based on a near-miss.

We will obviously improve our chances of success by looking for near-miss solutions for boards that cannot be solved directly. We can further improve the likelihood of success by considering "far-miss" solutions as well. Suppose we have conducted a near-miss search on the 4-family of a board, and no solution has been located. Rather than extend the near-miss search into the 5-family or 6-family (increasing the size of the near-miss search space exponentially with each expansion), we will apply a different modification to the initial board in order to arrive at a "far-miss" search space.

Given the initial board configuration, we generate a random path of 10 legal moves and apply that move sequence to the initial board. The resulting board configuration will now serve as the basis for a "far-miss" search. We first apply the case-base search to the resulting far-miss board, and if this search fails, we then apply a 4-family search to the far-miss board with the hope that our luck will be better in this new search space neighborhood. However, if this 4-family also fails, we can apply the same far-miss heuristic to our current board and continue to produce new 4-families as needed in a recursive manner. Eventually a solution will be found if we persist with enough far-miss searches.

Note that both near-miss and far-miss solutions involve modifications of the initial board before the case base can provide a solution path. It is therefore necessary to record the sequence of moves made during either a near-miss or far-miss board modification and append those moves to the move sequence provided by the case base. In general, we add an extra 1-4 moves to a solution path when the solution is based on a 4-family search, and we add an extra 10 moves to a solution path when the solution is based on a far-miss random walk modification. Because the far miss heuristic is applied recursively, the random walk moves accumulate. If three random walks are required before a solution is found, we must add 30 moves to the solution path (along with 1-4 moves if the solution is based on a near-miss of the far-miss).

2.4 Learning

The program we have described thus far does not learn. However, it is trivial to modify this program so that it can benefit from previously solved problems when those problems involve a near-miss or far-miss solution. In those cases, we generate a partial path of moves during board modification which must be appended to the solution path found in the case base. Suppose, for example, we generate a partial path of 23 moves derived from two random walks and a 4-family search. Further suppose that the solution path located for that modification picked up the last 15 boards in a case base solution of length 20. We can now generate a new case for the case base by appending the 23-move partial path with the 15-board subsequence from an existing case. Our new case contains 39 boards (23 moves correspond to 24 boards) and duplicates the

[3]On average, a 4-family contains 32.5 boards. See the implementation notes for more information about the generation of this search space.

last 15 boards from the existing case on which it is based.

In this manner we can augment the case base with new cases whenever solutions have been found that depend on board modifications. As the case base grows, we can expect to locate solutions after fewer searches of the case base. On the other hand, we should understand that any advantage gained by reducing the number of searches needed is also offset to some extent by the increased complexity of searching a larger case base. Implementation factors will determine whether or not it is more efficient to run with or without a learning capability.[4] In any case, a learning capability will guarantee immediate solutions to problems that have been previously solved since we are effectively remembering the specific solution path obtained.

3 Program Performance

We have implemented the program described above in Common LISP and run a number of experiments on a TI Explorer. Thirty test boards were generated by taking random walks of variable lengths away from the targeted final state. We will refer to these problem boards as P1-P30. Twenty solutions for an initial case base were generated in the same manner. We will refer to this case base as CB-20. The path lengths of the case base solutions in CB-20 range from 21 to 115 boards with an average length of 50 boards. The system monitors the number of case base searches it conducts for each problem, and terminates its processing if no solution is found after 200 searches. It is therefore possible for the system to fail to solve problems.

We have conducted test runs on P1-P30 with both a learning capability and no learning in place. We have monitored the number of searches required to solve each problem and the length of the solutions produced. Figures 3 and 4 show how the system performed on average during four independent test runs on the P1-P30 corpus with and without any learning. The number of failures refers to how many problems failed to be solved after 200 case base searches.

ave. no. of searches/problem	ave. solution length	no. of failures
60	43	3
72	44	2

Figure 3: Test Results without Learning on P1-P30 Using CB-20.

[4]Our implementation does benefit from the learning capability - comparative run times will be discussed in the next section

ave. no. of searches/problem	ave. solution length	no. of failures
47	40	1
45	41	0

Figure 4: Test Results with Learning on P1-P30 Using CB-20.

In both of the trials where learning was enabled, exactly 8 of the 30 problems were solved by accessing learned cases. Because the search algorithm is designed to exhaust all solutions in the original case base before considering learned solutions, these problems would have required additional searches if no learning had taken place. This accounts for the 30% drop in the average number of searches required. The slight drop in path length (a 7% reduction) reflects a drop in the average case base path length as the case base grows via learning. At the end of the two learning trials, the average case had decreased in length from 50 to 45 (a 10% reduction). It makes sense that the path lengths for solved problems might reflect the average solution lengths within the case base. However, we cannot always expect a reduction in average path lengths as a result of learning. In section 6 we will see how learning can increase the average path lengths of the case base under certain circumstances.

As noted earlier, the smaller number of searches required under learning need not translate into improved runtimes due to the overhead costs of searching a growing case base. Depending on the implementation, this overhead could counterbalance or even overwhelm the apparent advantage of fewer searches. To determine how our implementation weighs this trade-off, we conducted one comparison of runtimes over P1-P30 using CB-20. In two additional trials (not reported in figures 3 and 4), we found that the learning capability reduced the overall runtime by 15%. Since the effects of learning depend on how much learning has taken place, we then increased the size of our test corpus to see how runtimes are affected as experience accumulates.

In order to better understand the long-term effects of learning, we devised a test corpus consisting of the original 30 problem boards (P1-P30) followed by 30 new problem boards (P31-P60) for a total test set of 60 problems. In one run we collected data for P31-P60 without any learning throughout. In another experiment we ran the entire 60-board test corpus with learning enabled and collected data for the last 30 problem boards of this run (P31-P60). Comparing these two runs, we found that learning reduced the number of case-base searches performed by 39%. Learning also reduced the average length of a solution by 33%. Although the average CPU time required for each case-base search increased by 22% when learning was enabled, the reduction in searches was able to counter this overhead significantly: the total runtime for P31-P60 was reduced by 26% when learning was enabled.

It appears that the more experience we add to our case base, the better the overall performance of the system. We are less likely to experience search failures with a larger case base, and learning reduces (1) the number of case-base searches we need to conduct, (2) the average path length of a solution,[5] and (3) the total runtime required to find solutions. Furthermore, the more we learn, the more pronounced these improvements become.

[5]See section 6 for a discussion of sufficient conditions associated with this reduction.

4 Trading Optimality for Efficiency

By experimenting with the specific heuristics used in the near-miss and far-miss searches, we can alter the performance of the system in a variety of ways. For example, we arbitrarily chose 10 as the path length for random walks used by the far-miss modification. We didn't want a number that was too large since that would increase the average path lengths of our solutions, yet we needed a number that was large enough to take us into a new part of the search space. A random walk of length 10 is usually sufficient to guarantee a completely new search space when the near-miss search is based on a 4-family. But a random walk of length 8 used in conjunction with a 3-family near-miss search might prove to be just as effective. Extensive experimentation is needed to determine an optimal combination of near-miss and far-miss heuristics.

We can also improve on the system's performance by experimenting with different filtering operations during the near-miss search. For example, we ran some experiments using a filter that screened out all boards that did not have an index equal to 8 or 9. According to a distribution study of the adjacency-index, fully 51% of the boards in CB-20 are indexed by the values 8 or 9 (A complete distribution of adjacency indices is described in Section 5). So a filter that screens out all other indices will be throwing out roughly half of the available boards in any given 4-family (assuming this case base distribution is representative of the overall board distribution).[6]

Interestingly, the application of this filter reduces the number of required searches. The application of the filter forces the system to conduct fewer searches per 4-family. As a result, we are likely to examine more far-misses (take more random walks) during the overall search process. Figures 5 and 6 summarize the results of four independent test runs using the 8-9 filter, twice with a learning capability, and twice without. These trials were run using CB-20 and the P1-P30 test corpus.

ave. no. of searches/problem	ave. solution length	no. of failures
47	55	0
53	59	0

Figure 5: Filter Test Results without Learning on P1-P30 Using CB-20.

[6]Glancing over the output traces of our test runs, this percentage for the 8-9 filter looks about right, although we have made no attempt to verify this statistically.

ave. no. of searches/problem	ave. solution length	no. of failures
34	48	2
42	53	0

Figure 6: Filter Test Results with Learning on P1-P30 Using CB-20.

Comparing Figure 5 with Figure 6 we see a 24% drop in the number of searches when learning is enabled. This improvement is roughly consistent with the analogous improvement (30%) we saw in figures 3 and 4 where no filter was applied to the 4-families. In terms of absolute performance, the number of searches required under the 8-9 filter is much lower than the number needed without the filter. Without learning, the filter reduces the average number of searches by 24%. With learning, the filter reduces the average number of searches by 17%. Either way, the filter is beneficial if we want to minimize the number of searches (and runtime) required to solve a problem.

On the other hand, we do pay for the lower number of searches in terms of a longer average solution path. Without learning, the filter raises path lengths by 31%. With learning, the increase is 25%. If we want to minimize the length of our solutions, the 8-9 filter is definitely detrimental.

Comparing the various test runs, it is obvious that a trade-off between run-time efficiency and solution optimality can be exploited within this CBR framework. By trading near-miss solutions for far-miss solutions, we decrease the number of searches needed and increase the length of our solutions. If we were to enlarge the near-miss search to cover a 5-family instead of a 4-family, we could trade far-miss solutions for near-miss solutions and force the trade-off in the other direction.

5 What Makes Coarse Indexing Effective?

The indices we use to represent 8-puzzle boards are crucial to the effective operation of our program. The generality of our representation relies on the fact that only 12 indices are used to represent all possible board configurations. The effectiveness of our representation relies on the fact that these 12 indices are unevenly distributed across the set of all possible board configurations. Most importantly, the index used for the final state applies to only four 8-puzzle boards: the final state and the rotation group based on 90-degree rotations of the final state. Note that while reflections are index-preserving operations, we never encounter reflections of the final state because reflections do not preserve solvability for the 8-puzzle. The 9! possible placements of pieces do not all result in solvable 8-puzzle states: half of all possible placements are unsolvable

The adjacency index described in section 2.1 has a very convenient distribution for our purposes. While only 4 8-puzzle boards can be indexed by 0, the remaining 181,436 8-puzzle boards must distribute themselves among the other 11 indices. This means that the indices we use to describe a sequence of boards provide only a rough picture of the specific sequence at hand. It is useful to think of the indices as providing a rough path shape for a sequence of boards. Just as a projection of the plane reduces 2-dimensional shapes to 1-dimensional intervals, our indices will collapse a variety of distinct board sequences into the same "projected" path shape. If a sequence's path shape coincides with a subpath in the case base, we do not have a sufficient condition for a solution, but we do have a necessary one. At any step along the line we can "fall off" of the targeted projection, at which time the case base search must backtrack and try again.

Figure 7 shows how a specific problem board contacts an entry point in a case base which eventually takes us all the way home to the final state. To illustrate the power of coarse indices, we have reconstructed the original board sequence underlying the case base entry to show how much two board sequences can vary while preserving identical indices throughout. In this example, the two sequences converge to identical boards after 3 moves. (The problem board and the entry point board differ by 6 moves).

While we do not know exactly what the distribution profile of the adjacency index is over the complete set of 181,440 8-puzzle boards, we can hope to get some sense of the overall distribution by looking at index distributions for CB-20 and CB-30.[7] Figures 8 and 9 show how the adjacency index is distributed across our non-optimal and optimal case bases. Note that the number of 0's in each distribution corresponds to the number of cases in that case base.

While the two distributions do not have identical profiles, they do reveal some regularities that presumably reflect properties of the total distribution. Most striking is the fact that this is not a normal distribution. A normal distribution would be centered around the indices 6 and 7 (keeping in mind that the index cannot assume the value 1). CB-20 is centered around 8 and 9, while CB-30 is centered around 7 and 8. Is there some reason why the case base indices might not be representative of 8-puzzle boards in general? Yes, there is. But if we have to explain why the case base distributions might be off-center, we would be forced to predict a leaning in the *other* direction, since the case base paths all lead to 0. All legal 8-puzzle moves adjust the board index by -2, -1, 0, 1, or 2.[8] This means we cannot jump from a high index to 0 without moving through intermediate indices in a more continuous fashion. Therefore any path terminating in 0 must necessarily contain lower indices near the end of the path, regardless of the probabilities associated with those indices in the overall distribution of indices across all possible boards. In the event that these lower indices have small probability distributions in the larger scheme of things, our case bases will all elevate the frequency of these indices, just as we see 20 or 30 instances of boards indexed by 0 when we know that there are only 4 possible 8-puzzle boards mapping into this index. This might explain why the left side of the distribution levels off more gradually than the right side. The fact that our case base profiles favor indices on the high side of the spectrum suggests that there must be a marked preference for these indices in the overall distribution.

[7]CB-30 is an "optimal" case base containing 30 solution paths. CB-30 will be described in section 6.

[8]The one exception to this rule is a jump between 0 and 3, which occurs only because the adjacency index can never assume the value 1.

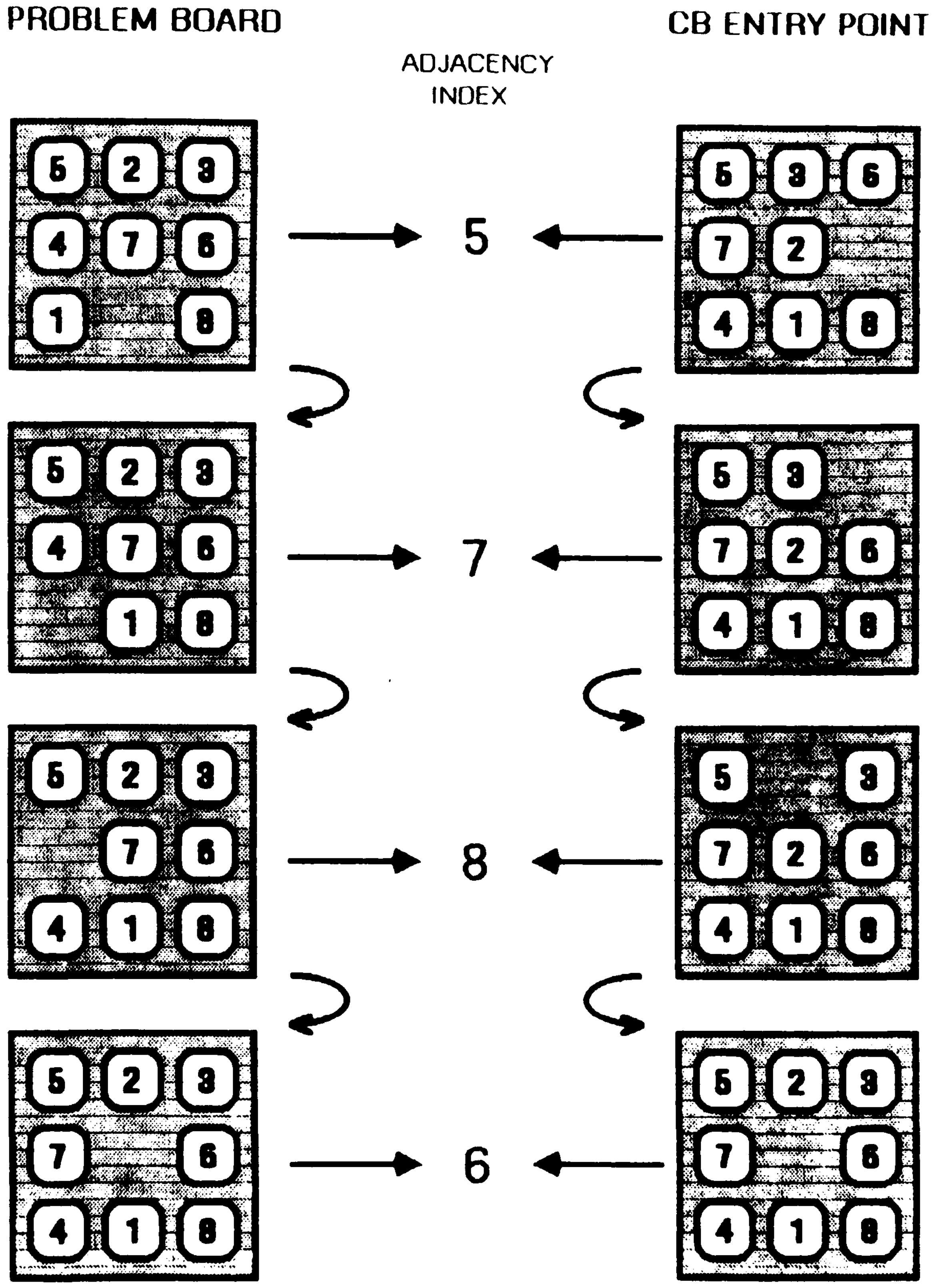

Figure 7: An Example of Coarse Indices

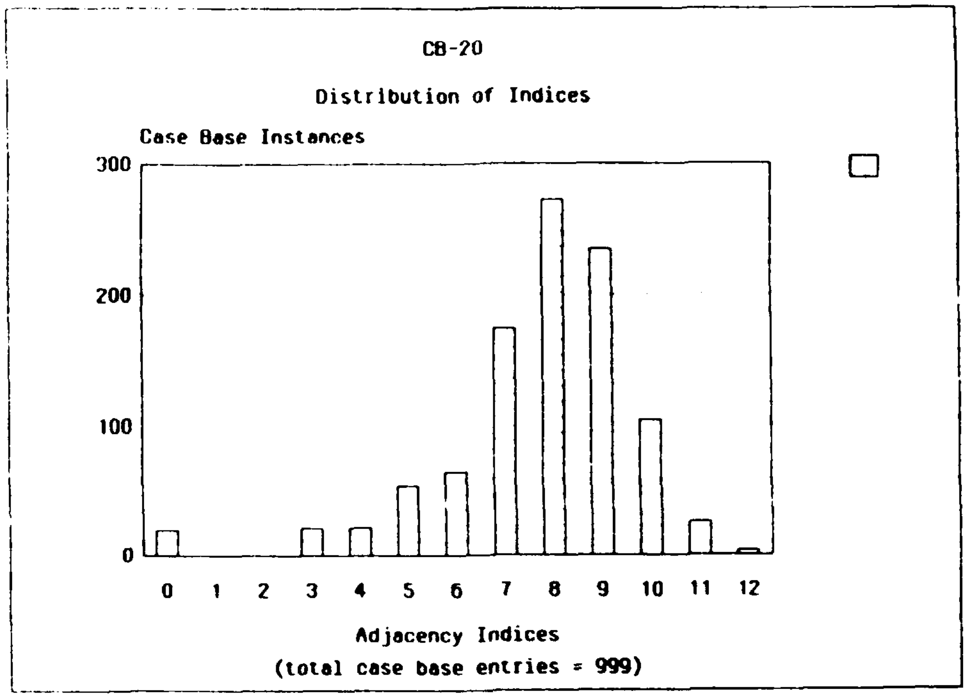

Figure 8: Adjacency Indices for CB-20

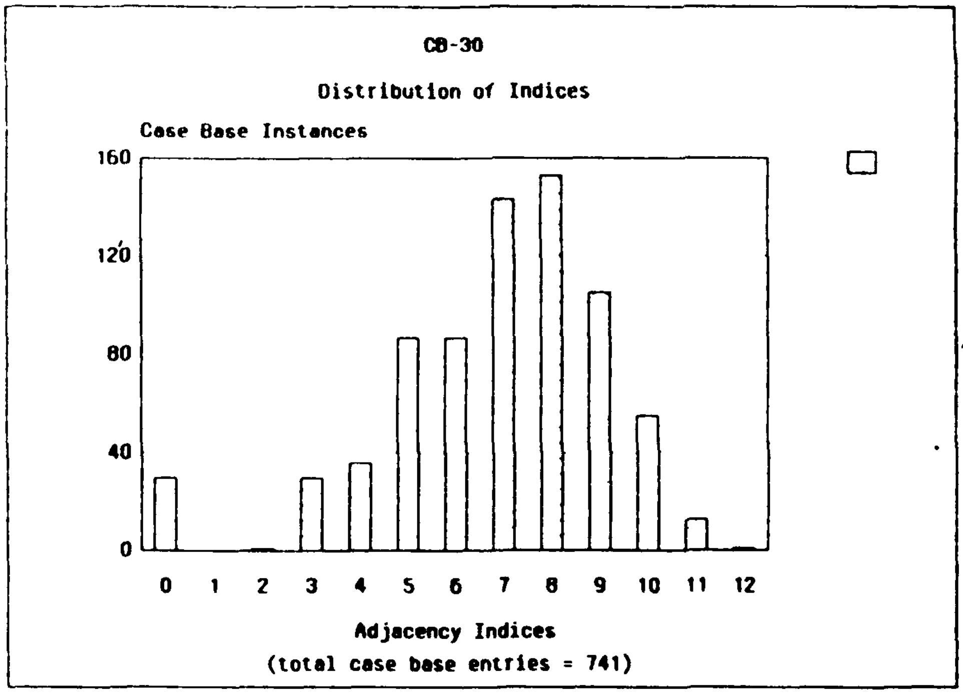

Figure 9: Adjacency Indices for CB-30

Regardless of what is happening with the overall distribution, the differences between the distributions for CB-20 and CB-30 carry an implication for the use of filters as described in section 4. In that section we reported on the effects of the 8-9 filter which was used in conjunction with CB-20. The effectiveness of that filter relied on the fact that 51% of all entries in CB-20 were either 8 or 9. If we consider the use of the 8-9 filter in conjunction with CB-30, we should note that only 35% of the entries in CB-30 were either an 8 or a 9. Given this difference between the two case bases, we would not expect CB-30 to work as well with the 8-9 filter as CB-20 did. A better filter for CB-30 would be a 7-8-9 filter since 54% of the CB-30 indices come from the set {7 8 9}.

In general, the utility of a filter relies on both the overall distribution of indices and the distribution profile of the current case base. The best filters utilize a heuristic that maximizes the likelihood of contact with case base entry points, while minimizing the subset of the total search space allowed to generate candidates for the case base search.

Aside from the issue of designing filters, there are some general design principles associated with the selection of an indexing scheme which will dictate the effectiveness of the case base search. Most obviously, it is crucial to find an indexing scheme which yields a manageable (small) number of states for the index representing the final state. The number of states described by the final state index will have to be handled by a finite table look-up routine that can massage the resulting final state projection into the true final state. The complexity of this post processing will dictate exactly what constitutes a manageable number of states.

Apart from the number of states we have associated with the terminating index, it is important to select an index whose distribution profile narrows dramatically as we approach the final state. It should be easy to establish contact with the case base initially, but progressively harder to maintain contact as we move further and further away from the entry point. Since it may not always be possible to derive, or practical to compute, a complete index distribution for a large search space, we expect that empirical techniques for comparing alternative indexing schemes will prove to be more illuminating than theoretical guidelines that presume knowledge of the complete index distribution.

6 Optimizing the Case Base

One way to optimize performance is by optimizing the case base. While it is easy to believe that some case bases will be better than others, it is not so obvious how to construct a good case base. Is it better to restrict the case base to cases that encode optimal solutions (minimal length) to initial boards? Is it better to include longer, non-optimal cases? There must be some ceiling on the size of a case base beyond which the costs of search outstrip the advantages of search space coverage. When does that line get crossed?

In an effort to answer some of these questions, we constructed a new case base (CB-30) of 30 problem solutions that are optimal in terms of the number of moves needed to reach the final state [Utgoff & Saxena, 1987b]. The average path length for this case base is 24.7 (as opposed to 50 for our first case base of 20 solutions). Utgoff has shown that the average path length for optimal solutions to the 8-puzzle is 21.97 moves (or 22.97 boards), so this case base is very close to the average complexity over all possible 8-puzzle boards. Our 30 boards were generated randomly, so there was no reason to expect perfectly average complexity. It is interesting to note that according to Utgoff's analysis, the longest optimal path for an 8-puzzle solution is 31 moves (or 32 boards).

Although CB-30 contains 30 solutions, it is actually smaller than CB-20 since the path lengths are substantially shorter in the optimal case base. The total number of boards encoded in CB-30 is 741 as opposed to 999 in CB-20. This means that the optimal case base provides fewer opportunities for a match than the original case base, so there is some small disadvantage associated with CB-30 as far as the case base searches are concerned.

We ran four trials with CB-30: two test sessions without learning and two sessions with learning enabled. For our test corpus we used P1-P30 (the same problem set used for most of our other experiments). The results of these trials are summarized below:

ave. no. of searches/problem	ave. solution length	no. of failures
96	32	7
113	32	11

Figure 8: Test Results Without Learning on P1-P30 Using CB-30.

ave. no. of searches/problem	ave. solution length	no. of failures
73	37	3
76	35	3

Figure 9: Test Results With Learning on P1-P30 Using CB-30.

We see from these results that learning decreases the number of case base searches by 29%. When these same tests were run on the original case base we saw a 30% reduction in the number of searches executed. From this comparison we conclude that the effects of learning are not sensitive to the optimality of the case base. On the other hand, learning under CB-30 does increase the path length of solutions by 12.7%. When these same tests were run on CB-20 we saw a 7% reduction in path lengths. This is not surprising since an optimal case base represents a lower bound on the length of problem solutions. Any solution discovered by the system involving near misses or far misses is not likely to be an optimal solution, so we should expect the average path length of the case base to increase with learning and result in subsequently longer solution paths.

In terms of absolute comparisons, the number of searches executed under the optimal case base is significantly greater than the number of searches executed under CB-20. Without learning, we conducted 58% more searches. With learning, we conducted 62% more searches. Some of this increase can be explained by the smaller number number of boards available in CB-30, but we suspect there would still be an increase in the number of searches required if the experiment were conducted with an optimal case base of comparable size. Since the distribution of optimal solutions for all 8-puzzle boards ranges from 0 to 32 with an average solution length of 23, we do not have a uniform distribution of solutions over the [0,32] interval. Since the "average" solution is closer to the high end of the interval, it will be harder to match boards into the case base at points closer to the final state. This means that a greater percentage of an optimal case base is harder to match into than would be the case for a non-optimal case base. A study of the case base entry point distributions would confirm this prediction.

The higher number of search failures experienced in these test runs reflects the higher number of case base searches executed. The more searches we need to execute, the more likely we are to exhaust the ceiling of 200 searches before a solution is found.

If we look at the average path lengths for solutions, we find that the solutions found under the optimal case base are indeed better than the solutions obtained under our non-optimal case base. Without learning, the path lengths are reduced by 26%. With learning, the path lengths are reduced by 11%. The shortest solutions are obtained using an optimal case base without learning, but even under those conditions we cannot expect to generate optimal solutions. The CBR approach allows us to approach optimality in solving problems, but there is no way to guarantee optimal solutions within this framework.

However, it is possible to optimize the performance of our system more than we have. Our current implementation conducts a case base search by looking for a legal path that ends with a 0 adjacency index, and stopping with the first such path it can find. We could redesign the search algorithm to collect all possible solution paths and return only the shortest one in the event that more than one solution is found. If it is important to maximize optimality, the additional overhead of such an exhaustive case base search would be offset by a lower path length average than we could obtain in the test runs described here.

In all of our test runs thus far, we have seen solution paths that are either longer or shorter than the average path length of the corresponding case base. If we are running with the learning capability enabled, these tendencies toward longer or shorter solutions translate into a tendency for the case base to increase or decrease its average path lengths. Under what conditions can we expect a case base to stabilize in terms of average path length? As we noted before, the distribution of entry points into the case base is neither a uniform nor a normal distribution. This makes it difficult to say much about necessary conditions for stability without more information about the distribution. But we can make a few observations for the sake of at least developing some rough intuitions about the problem.

Suppose, for example, that all boards in a case base were equally likely to provide an entry point during a case base search (this is a bogus assumption but it provides an easy baseline situation). Suppose also that all solutions are obtained by exact matches into the case base -- we have no near miss or far miss solutions. Then the average path length of a solution will be (1/2)APL where APL is the Average Path Length of the case base. The same result would hold for a normal distribution of entry points as long as the distribution was centered around (1/2)APL.

In this scenario, a learning capability would initially reduce the APL of the case base, and eventually pull it down to half of its original value (assuming the distribution of entry points remains uniform throughout [0,APL] or normal about (1/2)APL). But we are working with highly unnatural assumptions. In an optimal case base, the distribution of entry points is centered about APL (and is probably not a normal distribution anyway). We have also made an assumption that renders learning useless since we did not allow any near miss or far solutions in the scenario. Each solution obtained by an exact match into the case base can do nothing but duplicate a path of indices already present in the case base. If we add new cases based on exact matches, we do nothing but make copies of cases we already had.

To make the learning capability useful, we must consider the effects of near misses and far misses. If we are working with a near miss based on an N-family, then each near miss solution will add between 1 and N boards to a partial solution path found in the case base. If we are working with a far miss solution based on a random walk of length W, then each random walk adds another W boards to the partial solution path found in the case base. In general, the length of a solution is between KW + PL and KW + N + PL where K is the number of random walks taken and PL is the path length of the partial path found in the case base. The value of PL is governed by the distribution of entry points within the case base, the number of searches conducted for a solution can be computed on the basis of K and N, and K is a variable that tends toward 0 in any system that incorporates learning. (K tends toward 0 since it generally becomes easier to match into a case base the larger that case base is.)

It follows that a case base will stabilize when the APL of the case base and KW + PL + N converge. For non-optimal case bases, we might make a rough approximation for PL as (1/2)APL in which case we want to see KW + N tend tend toward (1/2)APL. Using this rough rule of thumb, we could go back the learning trials we ran with the 8-9 filter (Figure 6) and without any filter (Figure 4) to see which of these conditions resulted in a more stable case base. The average number of random walks executed during the two learning trials with the 8-9 filter was 1.9 and the average path length for the case base after those trials was 52.4.
2(KW + N) = 46 which is 88% of the APL = 52.4.
The average number of random walks executed during the two learning trials without any filter was .96 and the average path length for the case base after those trials was 44.95.
2(KW + N) = 27 which is 60% of the APL = 44.95.

This rough analysis therefore leads us to conclude that the APL resulting from the 8-9 filter learning trials is more stable than the APL resulting from the learning trials with no filter. Furthermore, we can expect both APL's to drop with more training, although the 8-9 filter condition will result in less of a drop than the no-filter condition. This prediction serves to distinguish the two conditions in terms of their average solution paths since the 8-9 filter condition was giving us average solutions paths of length 50 and the no-filter condition was giving us average solution paths of length 40. In the long run, the no-filter condition will win out over the 8-9 filte condition if we want to minimize the length of our solutions.

A more accurate analysis of this sort would be possible if we knew more about the distribution of entry points in our case bases. We have made a very simple assumption about this distribution by equating PL with (1/2)APL, and the accuracy of our predictions is necessarily sensitive to the validity of this assumption.

7 Case Base Reasoning in General

As we have seen with the 8-puzzle, it is possible to design a heuristic search algorithm that limits search based on known solutions within the same search space. In principle, one could start with a single solution and grow a case base base from there (although it may take a long time for the performance of the system to achieve a reasonable level of competence). In domains where we have a weak model of relevant domain knowledge, the CBR approach offers an alternative to the knowledge-engineering bottleneck associated with rule-based systems.

In our 8-puzzle program we have seen a number of issues that concern all CBR systems: the representation of cases in a case base, indexing schemes for accessing the case base, the design of near-miss and far-miss search heuristics, and the integration of learning as a natural outgrowth of the CBR approach to problem solving. We have also seen how it is possible to manipulate trade-offs between optimality in solutions and efficiency in runtimes, as well as how the general question of optimizing a case base might be pursued. All of these concepts are paradigmatic aspects of the general CBR approach to reasoning and problem solving, although specific solutions and methodologies for finding solutions may vary across domains.

In particular, the design of a case base may demand more powerful techniques of knowledge organization than the 8-puzzle can ever begin to approach. In most CBR domains, the case base embodies a domain model that defines the domain in terms of common vs. extraordinary circumstances. For example, in the domain of legal reasoning, a powerful case base is one that includes exceptional "landmark" cases that operate as prototypical precedents [Rissland, 1983; Ashley, 1987]. When the problem solving domain is approached in terms of planful behavior and adaptive planning, we see a case base that emphasizes planning failure as a way of realizing a failure-driven memory model [Hammond, 1987]. There is no one single set of techniques for devising a case-based memory any more than there is one set of standard techniques for devising a rule-based memory or a frame-based memory.

In fact, one could argue that our CBR approach to the 8-puzzle contains no domain model for the 8-puzzle at all. Although this may appear to distinguish the 8-puzzle as a somewhat pathological candidate for case-based reasoning, there is a form of CBR that has been dubbed "memory based reasoning" (MBR) which is characterized by a lack of domain models as well [Stanfill & Waltz, 1986]. For example, two different MBR approaches to the problem of English word pronunciation have been devised [Stanfill & Waltz, 1986; Lehnert, 1987] and neither of these systems characterize the domain in terms of important precedents or pathological pitfalls. Instead, both approaches utilize a stochastic data base. The effectiveness of these systems is therefore dependent on acquiring a case base of sufficient size for the statistics to prove meaningful. Beyond that, there is no obvious way to optimize the case base or tune it for performance trade-offs. MBR systems also operate in domains that seem to lack opportunities for explanation as well as run-time problem solving.

The line between MBR systems and CBR systems is not well-defined. Our 8-puzzle program appears to side with the MBR tradition with respect to explanation, run-time problem solving, and the question of a domain model. On the other hand, the 8-puzzle can be tuned for various optimizations in a way that none of the current MBR systems can be manipulated. The ability to play with the case base in useful ways seems much closer to the CBR style of programming and design.

Although we have used the 8-puzzle to illustrate how the CBR paradigm can be used to limit heuristic search, most CBR approaches to reasoning are not dependent on an exhaustive generate-and-test algorithm. For most CBR systems, a relevant case is located on the basis of dimensions or indices that characterize the problem at hand. Near-miss and far-miss search heuristics are then used to retrieve additional cases that might shed further light on the current problem. The primary problem solving task is therefore one of combining the relevant features of multiple cases into a coherent assessment of or solution to the original problem. In the 8-puzzle we never confront the problem of combining multiple cases since our case base search assumes an "all or nothing" attitude toward the cases under examination.

At the same time, the 8-puzzle does provide a simple and largely accurate way to think about learning within the CBR framework. Each new problem solved can be fed back into the case base to expand the power of the system, and the internal representation underlying that solution requires no substantial revisions to be of use to the case base. This distinguishes the CBR notion of "learning from examples" from rule-based efforts to learn from examples, where a major part of the rule-based approach entails translating examples into the internal rule base of the system. Although augmenting a case base is a simple idea, the form of memory organization inherent in a given case base will determine exactly what it means to augment the case base. In the 8-puzzle, augmentation amounted to a single cons operation. If a discrimination net is used to organize the base base, we may become involved with a variety of similarity-based learning algorithms, including inductive learning techniques [Quinlan, 1986] or generalization techniques [Lebowitz, 1986].

We have also seen how crucial it is to design powerful techniques for memory representation to ease the problem of effectively indexing the case base. The concept of coarse indices is generally applicable to any domain characterized by a case base of sequences and a need to execute heuristic tree searches. There are naturally domains where the problem space cannot be characterized in those terms. We need to develop methodologies and techniques for empirical evaluation that will help us at this most foundational level of system design in the same way that techniques for knowledge engineering have evolved along side rule-based systems.

Many proponents of CBR argue that it provides a powerful alternative to rule-based technologies. An analogy between LISP and FORTRAN may be useful in trying to understand these claims. As the last decade has shown us, rule-based expert system technologies have resulted in commercially viable applications of great generality. People can be trained to design expert systems in a matter of weeks, much as non-computer scientists can be taught the basics of

FORTRAN. If expert systems are limited in some respects, it is nevertheless possible to write useful computer programs within the expert systems paradigm.

However, AI researchers who are looking ahead to the next generation of intelligent systems are thinking about issues that were never intended to be cornerstones of the expert systems technology. We want programs that can improve with experience, explain themselves in a natural manner, and operate interactively with users who want to optimize some aspects of system performance at the expense of others. Although efforts are underway to construct these capabilities within the expert systems framework, progress has been slow and the obstacles are substantial. It may be necessary to consider a radically different framework if we truly wish to pursue these visionary capabilities.

Although it is new and relatively unknown, CBR framework appears to lend itself to learning, explanation, and optimization in a very natural way. Moreover, case bases can be designed to exploit a variety of AI techniques already under development within the machine learning community. A case base can provide fodder for inductive inference, self-organizing generalization, or explanation-based reasoning techniques. The only constraint that limits us is the fact that a case base must contain cases. After that, we are free to experiment in any number of directions. This new-found freedom is frankly exhilarating in the same way that LISP felt to a generation raised on FORTRAN. We may not know exactly what to do with it, but the ultimate benefits justify further experimentation.

The question of learning is rapidly becoming a central issue of serious concern for people interested in large AI applications. Real world systems are difficult to design, and even harder to maintain if the knowledge base underlying the system is one that needs to expand with time. The "knowledge-engineering bottleneck" has jumped out of our conference proceedings and into the laps of software engineers who worry about the half-life of computer programs that were overloaded before their beta tests were completed. Everything is reducible to a Turing Machine, but that doesn't mean FORTRAN and LISP are equivalent in terms of system development.

CBR is only now beginning to emerge as a viable paradigm for intelligent systems design. The area is ripe for theoretical advances and exploratory applications. We must allow our imaginations to run freely within the confines of the CBR paradigm in order to fully understand the boundaries of this new approach to intelligent reasoning. An informal survey of my colleagues who are active in CBR research revealed that no one felt that the 8-puzzle lent itself to their techniques. Indeed, I doubt that their research will benefit from the program described in this paper. On the other hand, we might all be wise to to hesitate on the question of how general CBR really is. For those with a taste for self-reference, we should note that a meta-CBR program would be able to input a collection of known CBR programs and use that case base to devise a new CBR program for a new problem domain. We might not be able to do it today, but that shouldn't stop us from trying.

Acknowledgments

The author is indebted to a number of people who have contributed to this work in one way or another. Edwina Rissland's long standing research program on legal reasoning sparked my initial interest in CBR. Paul Utgoff generated the optimal case base used for the experiments described in section 6. And last but not least, my graduate students have provided useful feedback throughout: Steven Bradtke, Don Ferris, David Haines, Tim Howells, Kishore Swaminathan, and Bob Williams. The computer implementation and experiments described here materialized shortly after ONR placed a TI Explorer in my office through the University Research Initiative Award # N00014-86-K-0764. Additional support has been provided by NSF Presidential Young Investigators Award NSFIST-8351863 and DARPA grant N00014-85-K-0017.

8 References

Alterman, R. (1986). An Adaptive Planner. *Proceedings of the Fifth National Conference on Artificial Intelligence.* (pp. 65-69). Philadelphia, PA: Morgan Kaufmann.

Ashley, K.D. (1987). Modelling Legal Argument: Reasoning with Cases and Hypotheticals. Ph.D. Thesis. Department of Computer & Information Science, University of Massachusetts, Amherst, MA.

Bariess, E., & Porter, B. (1987). Protos: An Exemplar-Based Learning Apprentice. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 12-23). Irvine, CA: Morgan Kaufmann.

Gaschnig, J. (1979). A Problem Similarity Approach to Devising Heuristics: First Results. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.* (pp. 301-307). Tokyo, Japan: Morgan Kaufmann.

Hammond, K. Explaining and repairing plans that fail. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* (pp. 109-114). Milan, Italy: Morgan Kaufmann.

Hammond, K. (1986a). CHEF: a model of case-based planning. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.* (pp. 267-271). Tokyo, Japan: Morgan Kaufmann.

Hammond, K. (1986b). Learning to anticipate and avoid planning problems through the explanations of failures. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.* (pp. 556-560). Tokyo, Japan: Morgan Kaufmann.

Kolodner, J., Simpson, R. and Sycara-Cyranski, E. (1985). A process model of case-based reasoning in problem solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 284-290). Los Angeles, CA: Morgan Kaufmann.

Korf, R., (1985). Macro-Operators: A Weak Method for Learning. *Artificial Intelligence 26.* 35-77.

Laird, J., Newell, A., & Rosenbloom, P. (1987). SOAR: An Architecture for General Intelligence *Artificial Intelligence, 33*, 1-64.

Laird, J., Rosenbloom, J., & Newell, A. (1984). Towards Chunking as a General Learning Mechanism. *Proceedings of the Fourth National Conference on Artificial Intelligence.* (pp. 188-192). Austin, TX: Morgan Kaufmann.

Laird, J., Rosenbloom, J., & Newell, A. (1986). Chunking in SOAR: the Anatomy of a General Learning Mechanism *Machine Learning, 1*, 11-46. Morgan Kaufmann.

Lebowitz, M. (1986). Concept learning in a rich input domain: generalization-based memory. *Machine Learning* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Lehnert, W. (1987). Case-based problem solving with a large knowledge base of learned cases. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 301-306). Seattle, WA: Morgan Kaufmann.

Nilsson, N. J. (1971). *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning 1*, Morgan Kaufmann.

Rissland, E., (1983). Examples in legal reasoning: legal hypotheticals. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* (pp. 90-94). Karlsrouhe, W. Germany: Morgan Kaufmann.

Simpson, R. (1985). *A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation.* Ph.D. Thesis. Technical Report No. GIT-ICS-85/18, School of Information and Computer Science, Georgia Institute of Technology. Atlanta, GA.

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning *Communications of the ACM, 29*, 1213-28.

Sycara, E.P. (1987). Resolving adversarial conflicts: an approach integrating case-based and analytic methods Ph.D. Thesis. Technical Report No. GIT-ICS-87/26, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.

Utgoff, P., & Saxena, S. (1987a). Learning a Preference Predicate . *Proceedings of the Fourth International Workshop on Machine Learning.* (pp. 115-121). Irvine, CA: Morgan Kaufmann.

Utgoff, P., & Saxena, S. (1987b). A Perfect Lookup Table Evaluation Function for the Eight-Puzzle COINS Technical Report 87-71. Department of Computer and Information Science, University of Massachusetts.

Appendix A: Implementation Tricks

1. The move generator used during tree traversal does not allow us to reverse the previous move. This significantly reduces the amount of backtracking needed during the case base search.

2. Similarly, when the 4-families are generated for the near-miss search, we do not use any moves that reverse the previous move. This prevents us from generating many duplicate boards within each family.

3. When near-miss and far-miss modifications are appended to a solution path we collapse any subsequences within the resulting move sequence that contain move reversals. This minimizes the length of new cases added to the case base during learning.

4. The set of all possible paths supported by the case base is represented by 12 separate trees, one for each root index. These case base trees are expanded only as needed using delayed evaluation, and unexpanded copies of the trees are frequently substituted for partially expanded trees in order to minimize the cons-space used.